UNIDAD 2: USO DE ESTILOS: PARTE 2: CSS AVANZADO

LINKS DE APOYO

https://www.eniun.com/tipos-diseno-web-cual-utilizo-responsive/
https://lenguajecss.com/css/
https://www.w3schools.com/css/
http://desarrolloweb.dlsi.ua.es/libros/html-css/ejercicios
https://htmlcheatsheet.com/css/

ÍNDICE

PARTE 2: CSS AVANZADO

1. UNIDADES DE MEDIDA

1.1. UNIDADES ABSOLUTAS

Se visualizan siempre igual independientemente del dispositivo.

px	Píxeles
cm	Centímetros
mm	Milímetros
in	Pulgadas (1 pulgada = 2.54 cm)
pt	Puntos (1 pt = 1/72 pulgadas)
pc	Picas (1 pica = 12 puntos)

1.2. UNIDADES RELATIVAS

Se ajustan a cada dispositivo ya que **dependen de la resolución de cada pantalla**.

%	relativo al elemento padre
em	Relativo al tamaño de la fuente actual Ej: **2 em =** 2 veces el tamaño de la fuente actual
vh vw	**Relativo al viewport (área visible según el dispositivo)** **1vh** = **1%** de la altura del viewport **100vh** =**100%** altura del viewport
fr	Se utiliza en **Grid** Layout Representa una **fracción** del espacio disponible en un contenedor

Aunque el **píxel** se considera **absoluta**, su **tamaño físico puede variar** según la densidad de píxeles por pulgada (PPI) del dispositivo.

En dispositivos con una mayor densidad de píxeles, como pantallas Retina, un píxel puede ser más pequeño en términos físicos, lo que resulta en una apariencia más nítida y detallada.

1.3. USOS DE UNIDADES

- Es **recomendable usar unidades relativas** en la medida de lo posible,
  - ya que mejora la accesibilidad
  - permite la adaptación fácil a cualquier medio.
- Por tanto, el uso de **medidas absolutas queda descartado**

Píxeles (px)

Representa un **punto** en la pantalla.

Se usa para **tamaños fijos** y proporciona control preciso sobre el diseño.

Aunque es recomendable utilizar unidades relativas en la medida de lo posible, el píxel sigue siendo una opción para un **diseño estático** o un control exacto sobre el tamaño de los elementos.

Porcentaje (%)

Representa una proporción del tamaño del elemento **padre**.

Es útil para hacer **diseños fluidos y responsivos** teniendo en cuenta la **relación de los elementos con su contenedor padre**.

Unidad em

Es útil para establecer **tamaños proporcionales al tamaño de fuente.**

W3C, recomienda el uso de **em** para indicar el **tamaño del texto**.

El tamaño de los ems se establece en base al tamaño que tenga definido el navegador.

Usualmente **el tamaño de una fuente por defecto en los navegadores es de 16px**.

Por tanto, **16px = 1em**

px	em	%
12	0,750	75
14	0,875	87,5
16	**1,000**	**100**
18	1,125	112,5
20	1,250	125

Existe **rem que** se basa en el **tamaño de fuente del elemento raíz** (generalmente el tamaño de fuente del elemento HTML).

Viewport Width (vw) y Viewport Height (vh)

Representan un porcentaje del ancho y alto de la **ventana del navegador.**

Son útiles para **diseños responsive basados en el tamaño de la pantalla**.

Flexible Grid Units (fr)

Se utiliza en Grid Layout

Representa una **fracción** del espacio en un contenedor.

Es útil para distribuir el espacio disponible entre elementos **flexibles.**

https://codepen.io/profe-ana/pen/dPGRJQo

2. TIPOS DE DISEÑOS

2.1. DISEÑO FIJO

- Dispone de **medidas fijas** en **px** que no son modificadas para los distintos dispositivos.
- Por lo tanto, **no cumple las normas de un diseño web responsivo**.

https://codepen.io/Eniun/pen/yLLVJre

2.2. DISEÑO ELÁSTICO

Se definen las **dimensiones** de la página **con unidades «em»**.

- **Se adapta cuando cambia el tamaño del texto**, pero
- **NO se adapta a los cambios de tamaño de la ventana del navegador.**

https://codepen.io/Eniun/pen/bGGBeyP

2.3. DISEÑO LÍQUIDO

Normalmente se definen los tamaños mediante **porcentajes**.

- Se **adapta** a la ventana del dispositivo
- Perjudica la experiencia de usuario
- **NO permite controlar el diseño** (el diseño varía según el tamaño del dispositivo y no se puede diseñar al píxel para todos los tamaños).

https://codepen.io/Eniun/pen/BaaQzbE

2.4. DISEÑO FLEXIBLE

- Utiliza **min-width** y **max-width** para que las anchuras puedan adaptarse dentro de unos mínimos y máximos.

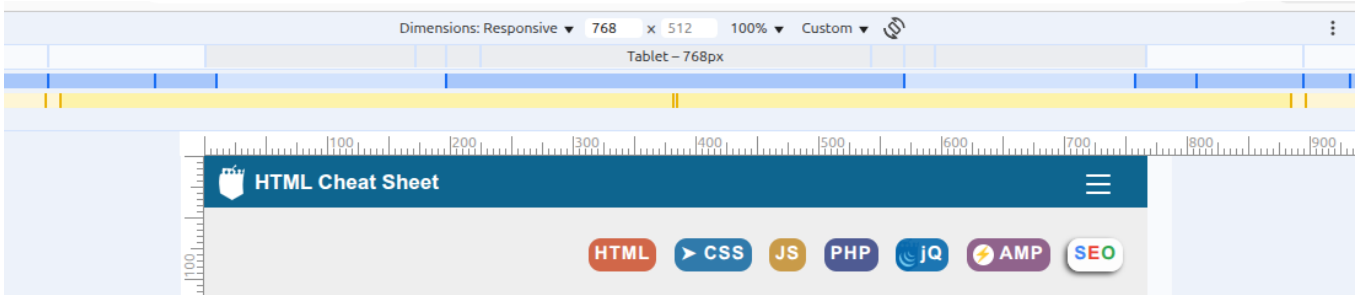
https://codepen.io/Eniun/pen/abNJZVr

2.5. DISEÑO WEB RESPONSIVO

- **Responsive (adaptable o *adaptative*)**
- Se basa en el uso de ***media queries***.
- **Define estilos condicionales con puntos de ruptura**

Breakpoint	Class infix	Dimensions
X-Small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

- **Ejemplo**:aplicar un estilo diferente en pantallas de ancho **superior** a 1200px, **inferior** a 1200px, e **inferior** a 700px: https://codepen.io/Eniun/pen/QWWGKLw
- Al inspeccionar la página podemos **ver los puntos de ruptura** para los distintos dispositivos



2.6. TIPO DE DISEÑO A UTILIZAR

- Lo ideal es utilizar un diseño web **responsivo**, preferentemente **no elástico** y **con mezcla de diseño flexible y líquido** según el contenedor.
- Lo habitual es declarar **múltiples *media query*** para adaptar el diseño a múltiples dispositivos.
- De esta manera ofreceremos la **mejor experiencia de usuario** y podremos considerar las **distintas medidas de los dispositivos**.

3. DISEÑO RESPONSIVE- MEDIA QUERY

Se basa en los conceptos:

- **viewport**: área visible que cambia según el dispositivo.
- **Breakpoint**: punto en el que cambian las propiedades css, normalmente atendiendo a **anchura**.
- **Media Query**: estilo css que se aplica de forma condicional atendiendo, normalmente al tamaño o resolución del dispositivo.

3.1. ETIQUETA VIEWPORT

- **Añadir en el head**
- Cuando trabajamos con **web responsive** es necesario **definir un viewport**, de lo contrario es muy probable que:
  - la página no lea correctamente los *media queries*
  - se vea en formato muy reducido, siendo necesario hacer zoom para ver el contenido.
- Permite definir los **parámetros** de visualización de una web en diferentes dispositivos.

Dispone de los siguientes **PARÁMETROS**:

- **width**: anchura virtual de la pantalla
- **height**: altura virtual de la pantalla
- **initial-scale:** la escala inicial del documento.
- **minimum-scale:** la escala mínima que se puede establecer.
- **maximum-scale:** la escala máxima configurable.
- **user-scalable**: si se permite al usuario hacer zoom.

EJEMPLO

- **width=device-width** conseguimos que el *viewport* **sea igual a la anchura real de la pantalla** del dispositivo, de modo veremos los píxeles reales.
- **initial-scale=1** conseguimos **definir la escala inicial** del documento para evitar transformaciones.
- **user-scalable=no** o con **maximum-scale igual al de initial-scale** conseguimos que el usuario **no pueda hacer zoom**, con lo que siempre se mantendrán las medidas indicadas la web.

Ejemplo no escalar:

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1">
```

EJEMPLO ESCALAR LA PÁGINA

Es interesante que el usuario pueda hacer zoom.

Para no limitar al usuario el uso del zoom: **no definir maximum-scale ni el user-scalable=no**.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

3.2. ESTILOS PARA DIFERENTES MEDIOS O DISPOSITIVOS

Medio	Descripción
all	Todos los dispositivos
print	Para documentos paginados y mostrados en vista de impresión
screen	Pantallas de ordenador
speech	Sintetizadores para navegadores de voz utilizados por personas discapacitadas

Hay tres **FORMAS DE INDICAR EL MEDIO**

1. <LINK>

media= "medio" que se aplica el estilo del archivo indicado en **href**.

```
<link rel="stylesheet" type="text/css" media="screen" href="style.css" />
<link rel="stylesheet" type="text/css" media="print" href="especialStyle.css" />
```

2. @MEDIA

@media nombre del dispositivo

Si los estilos se aplican a **varios medios**, se incluyen los nombres **separados mediante comas**.

```
@media print { body { font-size: 11pt; } }
@media screen { body { font-size: 12px; } }
@media { body { color: blue; } }
```

3. @IMPORT

Enlaza archivos CSS externos para cada medio.

```
@import url("estilosPantalla.css") screen;
@import url("estilosImpresora.css") print;
@import url("estilos.css");
```

3.3. MEDIA QUERIES

- Dan respuesta a las necesidades del diseño web ***RESPONSIVE***.
- Permite **definir estilos condicionales, aplicables a determinadas situaciones.**
- Permite establecer **estilos diferentes** para cada **ancho de pantalla**.

1. DEFINIR ANCHO MÁXIMO

Ejemplo: aplicar un estilo diferente en pantallas de **ancho superior a 1024px, inferior a 1024px e inferior a 480px,**

```
.click:after {
   content:"En pantalla grande";
}
@media (max-width: 1024px) {
   .click:after {
      content:"Tablet";
   }
}

@media (max-width: 480px) {
   .click:after {
      content:"Movil";
   }
}
```

Ejemplo: https://codepen.io/Eniun/pen/poogWZj

2. OPERADORES LÓGICOS

Se utilizan para comprobar una o varias condiciones.

AND

Cuando la pantalla tenga un ancho mínimo de 700px **y** la orientación de la misma sea horizontal (*landscape*).

@media (min-width: 700px) and (orientation: landscape) { ... }

NOT

Cuando no se cumpla la condición especificada

@media not screen and (monochrome) { ... }

3. OTRAS PROPIEDADES

Nombre	Descripción
width	Anchura del *viewport*
height	Altura del *viewport*
aspect-ratio	Relación de aspecto anchura-altura del *viewport*
orientation	Orientación del *viewport*
resolution	Densidad de píxeles del dispositivo
scan	Proceso de escaneo del dispositivo
grid	Si el dispositivo es grid o bitmap
update-frequency	Velocidad de actualización del dispositivo para modificar la apariencia del contenido
overflow-block	Cómo maneja el dispositivo el contenido que excede los límites del *viewport* a lo largo del eje de bloque
overflow-inline	Cómo maneja el dispositivo el contenido que excede los límites del eje *inline*
color	Componente de número de bits por color del dispositivo, o cero si el dispositivo no es a color
color-index	Número de entradas en la tabla de búsqueda de color del dispositivo, o cero si el dispositivo no usa una tabla
monochrome	Bits por píxel en el buffer de marco monocromático del dispositivo, o 0 si el dispositivo no es monocromático
hover	Si se puede posicionar el puntero sobre los elementos

https://www.w3schools.com/css/css_rwd_mediaqueries.asp

3.4. EJEMPLOS RESUMEN MEDIA QUERIES

Ejemplo1: cursos: https://codepen.io/Eniun/pen/PoKWdWe

Ejemplo2: blog con aside

4. FLEXBOX (1 DIMENSIÓN)

1. CARACTERÍSTICAS

- Ofrece un mecanismo para **dividir el espacio de la ventana, mediante filas y columnas.**
- **No** necesita utilizar los **posicionamiento** de cajas (static, relative, absolute, float…)
- Organiza los elementos utilizando **contenedores flexibles**.
- Es necesario **crear un contenedor padre** que configure las características de los elementos hijos.
- Al contenedor padre debemos declararle la propiedad **display:flex**

```
#container{
    display: flex;
}
```

EJEMPLO: diferencia entre display flex y float: https://codepen.io/profe-ana/pen/qEbXZjO

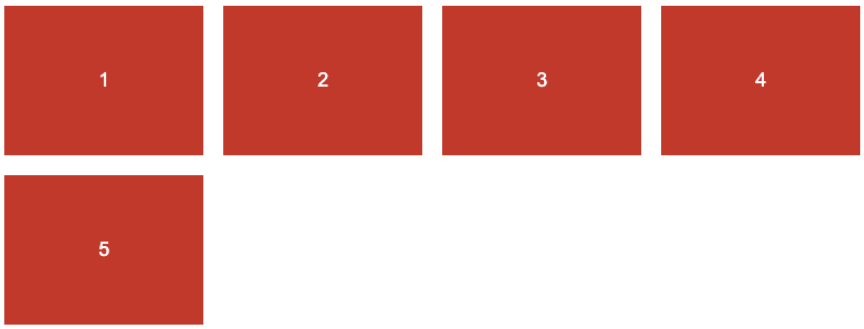
2. FLEX-WRAP

Indica qué elementos flexibles **se deben trasladar cuando no hay suficiente espacio** en el contenedor.
Sus VALORES son:

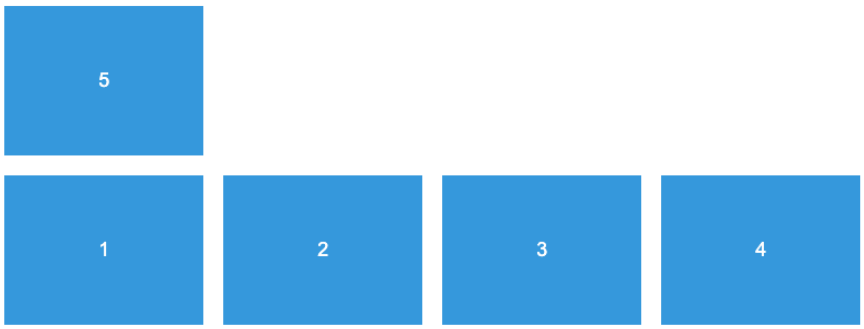
- **nowrap: valor por defecto:** los elementos **no pasan a la siguiente fila** y reducen su anchura.
- **wrap:** los elementos **pasan a la siguiente fila** y conservan su anchura.
- **wrap-reverse:** los elementos pasan a la siguiente fila, pero en sentido **inverso**.

flex-wrap: no-wrap;

flex-wrap: wrap;



flex-wrap: wrap-reverse;



Ejemplo: https://codepen.io/Eniun/pen/XWWVdGb

3. JUSTIFY-CONTENT

Define la justificación horizontal de los elementos hijos en un contenedor flexible.
VALORES:

- **flex-start**: posiciona los elementos a la **izquierda** del contenedor. Valor **por defecto**.

- **flex-end**: posiciona los elementos a la **derecha** del contenedor.
- **center**: **centra** los elementos en el contenedor
- **space-between**: añade un espacio idéntico entre los elementos. El primer elemento está a la izquierda, y el último, a la derecha.
- **space-around**: espacia de forma regular los elementos que no están alineados a la izquierda y a la derecha del contenedor.

justify-content: flex-start;

justify-content: flex-end;

justify-content: center;

justify-content: space-between;

justify-content: space-around;

Ejemplo: https://codepen.io/Eniun/pen/MWWreKY

4.ALIGN-ITEMS

Define alineación **vertical**

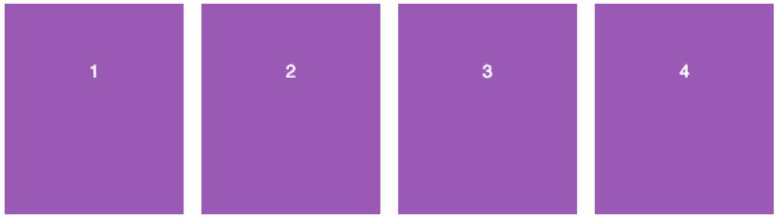
Primero se debe definir la propiedad **flex-direction: row**.

VALORES:

- **stretch**: los elementos se amplían verticalmente para ocupar toda la altura en el contenedor. Es el **valor por defecto.**
- **flex-start:** los elementos se colocan en la parte superior del contenedor.
- **flex-end:** los elementos se colocan en la parte inferior del contenedor.
- **center:** los elementos se colocan en el centro del contenedor.
- **baseline:** los elementos se alinean sobre la línea de base del texto.

```
#container{
    display: flex;
    flex-direction: row;
    align-items: stretch;
}
```

align-items: stretch;



align-items: flex-start;

align-items: flex-end;

align-items: center;

align-items: baseline;

EJEMPLO 1: align items: https://codepen.io/profe-ana/pen/XJXadYW
EJEMPLO 2: https://codepen.io/Eniun/pen/mddXYoR

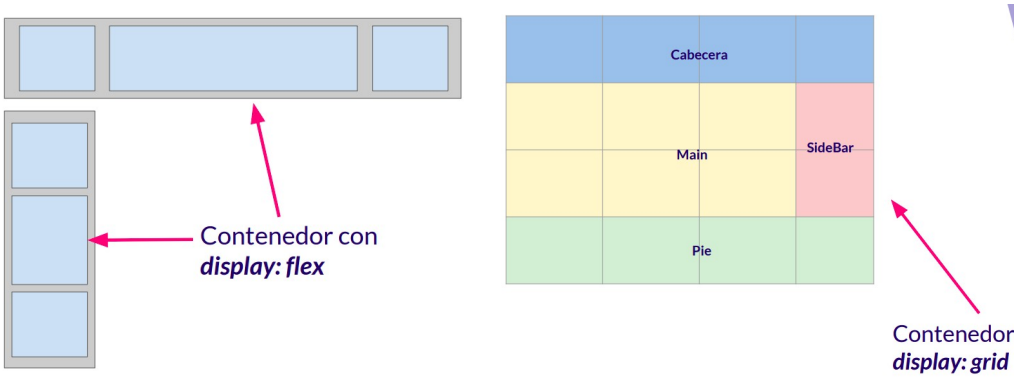
5. GRID (2 DIMENSIONES)

Permite maquetar contenido ajustándolo a **cuadrículas** (grids).

Gracias a CSS Grid y CSS Flexbox se pueden crear estructuras con menos código y de una forma más fácil que con los métodos tradicionales.

DIFERENCIA entre CSS Grid y CSS Flexbox

- **Flexbox** se creó para diseños de **una dimensión**, en una fila o una columna
- **Grid** se pensó para el diseño **bidimensional**, en varias filas y columnas



Para utilizar **CSS Grid**

- Definir un **contenedor padre** y en su interior los **ítems** que se necesiten.
- Se definen las propiedades, para el contenedor padre y para los ítems del interior.

DISPLAY

Valor	Descripción	Ejemplo
inline-grid	Cuadrícula en **línea** con respecto al contenido exterior	https://codepen.io/Eniun/pen/RwMxvON
grid	Cuadrícula en **bloque** con respecto al contenido exterior	

TAMAÑO DE COLUMNAS Y FILAS

Propiedad	Valor	Descripción	Ejemplo
grid-template-columns	[*col1*] [*col2*] …	Tamaño columna	.contenedor{ display: grid; **grid-template-columns: 50px 300px;** **grid-template-rows: 200px 75px;**}
grid-template-rows	[*fila1*] [*fila2*] …	Tamaño fila	

Los tamaños se pueden indicar en px, auto(se adapta al tamaño del contenedor) fr(fracción restante)

ESPACIADO ENTRE COLUMNAS Y FILAS

Propiedad	Descripción	Ejemplo
column-gap	Espaciado entre columnas	.contenedor{ display: grid; grid-template-rows: 200px; 200px grid-template-columns: 200px; 200px; 200px; **column-gap: 20px;** **row-gap: 10px;**}
row-gap	Espaciado entre filas	

EJEMPLO columnas distinto ancho con espaciados: https://codepen.io/profe-ana/pen/Poevopm
EJEMPLO completo de filas y columnas: https://codepen.io/profe-ana/pen/ExLzavN

GRID POR ÁREAS

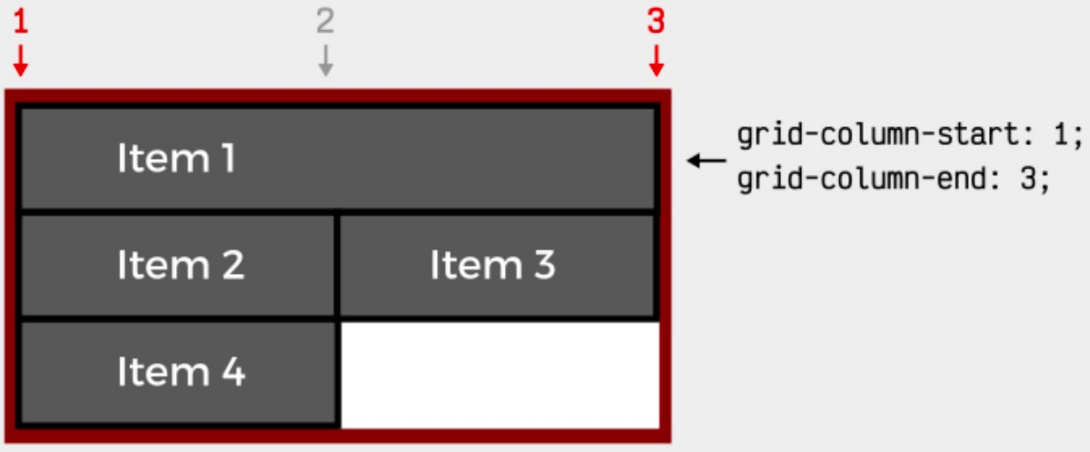
- Se indica el **nombre y posición** de cada **área** de una cuadrícula.
- En el **contenedor padre,** utilizaremos la propiedad **grid-template-areas** , donde se **especifica el orden** de las **áreas** en la cuadrícula.
- En cada **ítem hijo**, utilizamos la propiedad **grid-area** para indicar el nombre del área

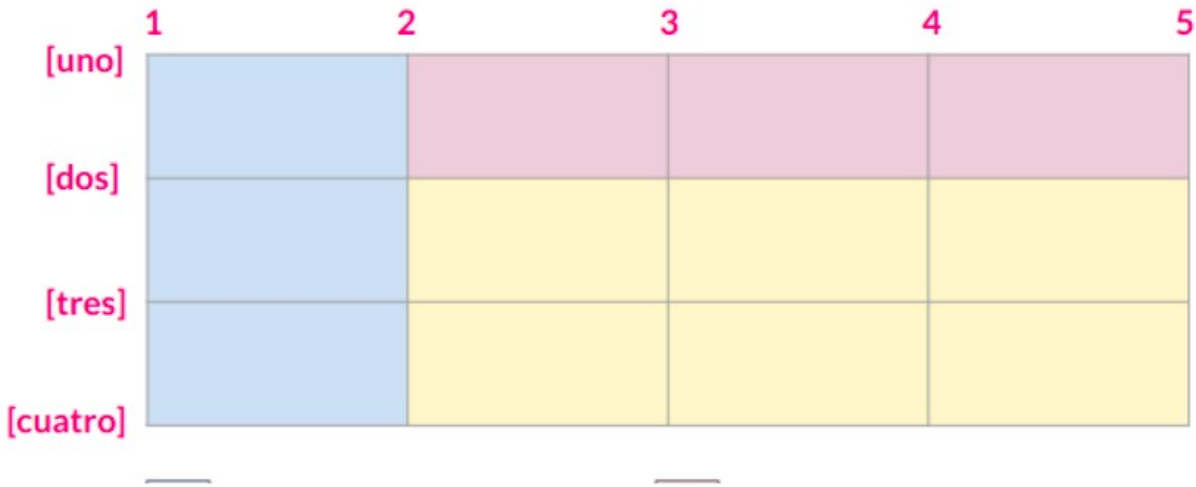
```
.container {
     display: grid;
     grid-template-areas: "head head head"
                          "menu1 main menu2"
                          "foot foot foot";
 }
.item-1 { grid-area: head; background: blue; text-align: center; color:aliceblue}
.item-2 { grid-area: menu1; background: red; text-align: center;color: azure;}
.item-3 { grid-area: main; background: green; text-align: center;color: beige;}
.item-4 { grid-area: menu2; background: red; text-align: center;color: azure;}
.item-5 { grid-area: foot; background: orange; text-align: center;color: chartreuse;}
```

EJEMPLO : https://codepen.io/profe-ana/pen/VwxOvZw

DISTRIBUCIÓN IRREGULAR DE CELDAS DEL GRID(combinar celdas)

Propiedad	Descripción
grid-column-start	Indica en que columna empezará el ítem de la cuadrícula.
grid-column-end	Indica en que columna terminará el ítem de la cuadrícula.
grid-row-start	Indica en que fila empezará el ítem de la cuadrícula.
grid-row-end	Indica en que fila terminará el ítem de la cuadrícula.






```
#azul {
background-color: blue;
grid-column-start: 1;
grid-column-end: 2;
grid-row-start: uno;
grid-row-end: cuatro;
}
```

```
#rojo {
background-color: red;
grid-column-start: 2;
grid-column-end: 5;
grid-row-start: uno;
grid-row-end: dos;
}
```

```
#amarillo {
background-color: yellow;
grid-column-start: 2;
grid-column-end: 5;
grid-row-start: dos;
grid-row-end: span 2;
}
```

Podemos juntar estas propiedades:

```
grid-column: start / end;
grid-row: start / end;
```

EJEMPLO DE DISTRIBUCIÓN IRREGULAR:
https://codepen.io/profe-ana/pen/ExLzVVM

Si no hemos especificado el área de los elementos de l GRID podemos indicar en el contenedor:

- **grid-auto-row:** Rellena primero las filas. Es la opción **por defecto.**
- **grid-auto-column:** Rellena primero las columnas. Es la opción **por defecto.**
- **grid-auto-dense:** Intenta rellenar primero los huecos por si viene elementos posteriores más pequeños. Hay que TENER CUIDADO porque puede provocar cambios de orden en los elementos.

MÁS INFORMACIÓN:
https://www.w3schools.com/css/css_rwd_grid.asp
https://lenguajecss.com/css/maquetacion-y-colocacion/grid-css/

6. FLEXBOX O GRID

Hay cosas que sólo puedo hacer con Flexbox.
Hay cosas que sólo puedo hacer con Grid.
CONCLUSIÓN: usar ambos según interese. Nada impide que un área de un contenedor GRID sea a su vez un contenedor FLEX.

7. CENTRAR HORIZONTAL Y VERTICALMENTE

7.1. CON FLEXBOX

Es el **método más útil** porque funciona para **todos los elementos que se posicionen dentro de un contenedor.**

Es necesario crear un contenedor con la propiedad **display: flex**.

```
div{
   height: 200px;
   background-color: #EEE;
   display:flex;
   justify-content: center;/*horizontal*/
   align-items: center; /*vertical*/
}
```

EJEMPLO: https://codepen.io/Eniun/pen/RwRyNav

7.2. CON TEXT-ALIGN Y LINE-HEIGHT

Para **únicamente textos(párrafos, encabezados**, etc)

```
div {
   height: 200px;
   background-color: #EEE;
   text-align: center;
}
div h1 {
   line-height: 200px;
}
```

EJEMPLO: https://codepen.io/Eniun/pen/OJXZPpJ

7.3. CON TRANSFORM

Se **puede alinear tanto vertical como horizontalmente imágenes y contenedores**.

No es válido para textos ya que no tiene en cuenta el tamaño de la letra.

```
div {
   height: 300px;
   background-color: #EEE;
   position: relative;
}
div img{
   position: absolute;
   left: 50%;
   top: 50%;
   transform: translate(-50%, -50%);
   -webkit-transform: translate(-50%, -50%);
}
```

EJEMPLO:https://codepen.io/Eniun/pen/MWeGYbp

7.4. ALINEAR TEXTO CON IMAGEN VERTICALMENTE

vertical-align admite los valores ***top***, ***middle*** y ***bottom***, entre otros valores.

```
img{ vertical-align: middle; }
```

EJEMPLO:https://codepen.io/Eniun/pen/ExyLaBO

7.5. ALINEAR TEXTO VERTICALMENTE EN UN DIV

vertical-align no se comporta de la misma manera con todos los elementos.

En este caso, haremos uso de **display: inline-table** y **display: table: cell**

```
div{
  height: 200px; width: 200px;
  background-color: #EEE;
  display: inline-table;
}
p{
  display: table-cell;
  vertical-align: middle;
}
```

EJEMPLO: https://codepen.io/Eniun/pen/NWbNYoN

7.6. ALINEAR IMAGEN HORIZONTALMENTE

text-align

```
.a{text-align: left;}
.b{text-align: center;}
.c{text-align: right;}
```

EJEMPLO: https://codepen.io/Eniun/pen/NWbNJWV

7.7. ALINEAR CHECKBOX CON ETIQUETA VERTICALMENTE

Cuando el tamaño del texto de la etiqueta no coincide con el tamaño del *checkbox*

vertical-align:middle

```
label, input{
   font-size: 28px;
   vertical-align: middle;
}
```

EJEMPLO:https://codepen.io/Eniun/pen/pobVJva

8. OVERFLOW

- Controla el **comportamiento del contenido que se encuentra en una caja o contenedor**.
- Podremos especificar si queremos **recortar un contenido**, **mostrar barras de desplazamiento** o **mostrar el contenido que excede de un elemento a nivel de bloque**.

Solo funciona sobre **elementos de tipo bloque con una altura definida**.

Valores

- **overflow: visible (default).** Los contenidos que se salen del elemento son visibles.
- **overflow: hidden.** Los contenidos que se salen del contenedor padre se ocultan y no se muestran barras de *scroll*. De esta forma se puede controlar el tamaño del elemento y su contenido.
- **overflow: scroll**. Se muestra una barra de *scroll* (horizontal y vertical) cuando los contenidos no caben en el contenedor o caja.
- **overflow: auto**.
    - El navegador es el que decide si se muestran las barras de *scroll* o si se extiende el contenedor.
    - Gracias a este valor nunca se permite que el contenido desborde al contenedor.
    - Si el contenido se sale por un lado sólo se muestra la barra de *scroll* de ese lado.

También existen las propiedades overflow-x y overflow-y.

EJEMPLO: https://codepen.io/Eniun/pen/poorvzK

9. EJEMPLOS

9.1. BOTÓN FIJO EN LA PARTE INFERIOR DERECHA

https://codepen.io/Eniun/pen/qBBZmmR

9.2. CONTENEDOR FIJO + CONTENEDOR VARIABLE O CON PORCENTAJE

Para conseguir un contenedor fijo y otro variable, utiliza función **calc()** de CSS

https://codepen.io/Eniun/pen/PooveoE

9.3. DISEÑO LÍQUIDO CONTENEDORES MEDIANTE PORCENTAJES CON BOX-SIZING

https://codepen.io/Eniun/pen/mddBGWV

9.4. FLOAT/ INLINE-BLOCK /INLINE

https://codepen.io/Eniun/pen/BaabKqZ

9.5. MEDIA QUERIES

https://codepen.io/Eniun/pen/PoKWdWe

9.6. FLEXBOX

https://codepen.io/Eniun/pen/mddXYoR

9.7. ALINEACIÓN CON FLEX

https://codepen.io/profe-ana/pen/YPwxWRJ

9.8. CENTRAR VERTICAL Y HORIZONTAL DE UN CONTENEDOR

https://codepen.io/Eniun/pen/JjjRyym

9.9. ALINEAR IMÁGENES CON TEXTO O BOTONES

https://codepen.io/Eniun/pen/MWWxyPP

9.10. EJEMPLO COMPLETO

https://codepen.io/Eniun/pen/OJJQdaj

9.11. OTROS EJEMPLOS

LISTA CONVERTIDA EN MENÚ HORIZONTAL: https://codepen.io/Eniun/pen/JjjGZQp

FORMULARIO BÁSICO: https://codepen.io/Eniun/pen/xxxEZwz

10. HERRAMIENTAS ÚTILES

CSS Validation Service, W3C

W3C es el encargado de estandarizar los estilos CSS y en su plataforma nos ofrece varias herramientas e información muy valiosa para la elaboración de nuestros desarrollos.
Entre las herramientas destacadas se encuentra el validador CSS que nos permite encontrar errores en nuestros ficheros.

Browserling

Browserling una página que nos permite probar cualquier web en los diferentes navegadores: Internet Explorer, Firefox, Chrome, Opera y Safari, en distintas versiones y resoluciones de pantalla.

Caniuse

Puedes ver los navegadores que soportan una determinada propiedad CSS o un elemento HTML5 en la página web caniuse.com

Extensión Autoprefixer para Visual Studio Code

Para ahorrar tiempo y facilitarnos la tarea de incluir los prefijos de las propiedades CSS que todavía no son estables podemos hacer uso de la extensión "Autoprefixer" en Visual Studio Code.

Patrones de gradientes

Podemos aplicar diseños muy trabajados creando **patrones** sin necesidad de cargar imágenes con mucho peso. Existen diversos artistas dedicados a hacer este tipo de diseños.
Algunos ejemplos de diseños de este tipo se pueden obtener en la siguiente plataforma:

- leaverou.github.io/css3patterns

10.1. REFERENCIAS WEB

- lenguajecss.com
- developer.mozilla.org/es/docs/Web/CSS
- w3schools.com/css/default.asp
- desarrolloweb.dlsi.ua.es/libros/html-css/ejercicios
- w3c.es
- desarrolloweb.dlsi.ua.es/libros/html-css/ejercicios
- en.wikipedia.org/wiki/Comparison_of_browser_engines_(CSS_support)
- escss.blogspot.com/2012/06/css-speech-module-css-hablado.html
- websitesetup.org/wp-content/uploads/2019/11/wsu-css-cheat-sheet-gdocs.pdf

10.2. CURSOS RECOMENDADOS

- https://openwebinars.net/academia/aprende/responsive-web-design/
- https://openwebinars.net/academia/aprende/flexbox-css-grid/